Testing-JUnit and TestNG

Testing with JUnit or TestNG

Various approaches to initializing your application for testing

For unit type tests, initialization isn't usually a problem, as unit testing only tests a small "unit" isolated and not interacting with all the other "units" which make up your application. But there are other tests which may require many units of your application working cooperatively, and in order to work properly, all these units must be initialized properly.

Create a method which runs once before any tests

Your application should be initialzed just once before the tests. There are annotations etc. which allow one to pick a method which will be run prior to any tests. For a TestNG example (similar to Junit 4):

```
package example.com.test;

import er.extensions.appserver.ERXApplication;
import java.util.Properties;

import org.testng.annotations.BeforeTest;

import com.webobjects.appserver.WOApplication;
import com.webobjects.foundation.NSBundle;

public class WOInit {
        static NSBundle MYBUNDLE = NSBundle.mainBundle();

        @BeforeTest  // init once
        public void initWO() {

                System.out.println("****InitWO starting...mainbundle is " + NSBundle.mainBundle().bundlePath());
                // somehow 'new' seems to init the static structures best
                example.com.app.Application myApp = new example.com.app.Application( ) ;
                //ERXApplication.primeApplication(NSBundle.mainBundle().bundlePath(), null, "example.com.app.
Application");
                //er.extensions.ERXExtensions.initApp(example.com.app.Application.class, new String[0]) ;
                System.out.println("****InitWO lists mainbundle:");
                MYBUNDLE.properties().list(System.out);
                Properties props = System.getProperties();
                System.out.println(props.toString());
                System.out.println("****InitWO ...Finished... ");

        }

}
```

There are various ways (some commented out) to initialize your application in the above example.

- example.com.app.Application myApp = new example.com.app.Application( ) ;
- ERXApplication.primeApplication(NSBundle.mainBundle().bundlePath(), null, "example.com.app.Application"); //(Wonder) or
- WOApplication.primeApplication(NSBundle.mainBundle().bundlePath(), null, "example.com.app.Application"); //(No Wonder)
- er.extensions.ERXExtensions.initApp(example.com.app.Application.class, new String[WO:0]) ;

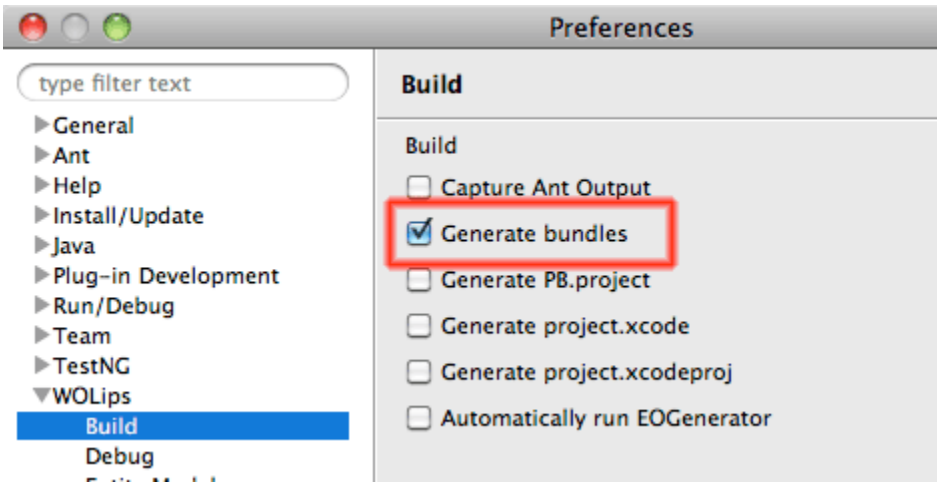
One of them may work best in your situation. It may be important for your application to find its main bundle properly, and all the goodies bound up in that main bundle.

Minor configuration in eclipse

To be able to test your business objects layer in JUnit or TestNG test cases, you should set the following parameters in your test case/test suite launch configuration depending on if you are using *normal* or bundle-less builds.

Normal builds

A *normal* build means that you have checked the build option *Generate bundles* within the WOLips preferences.

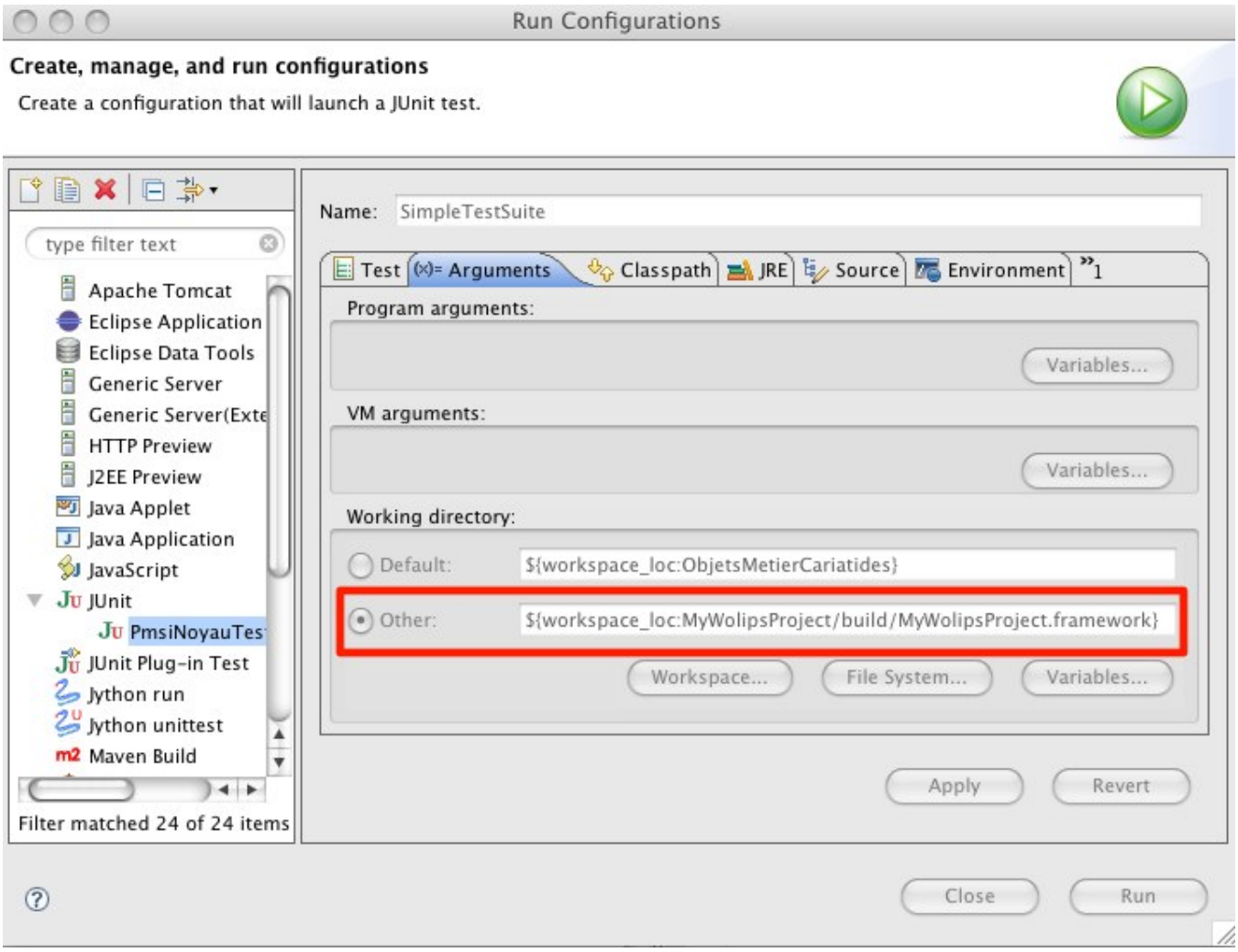


First in the *"Arguments"* tab, set your working directory to the build product of your project. For example, for a framework project the working directory should be:

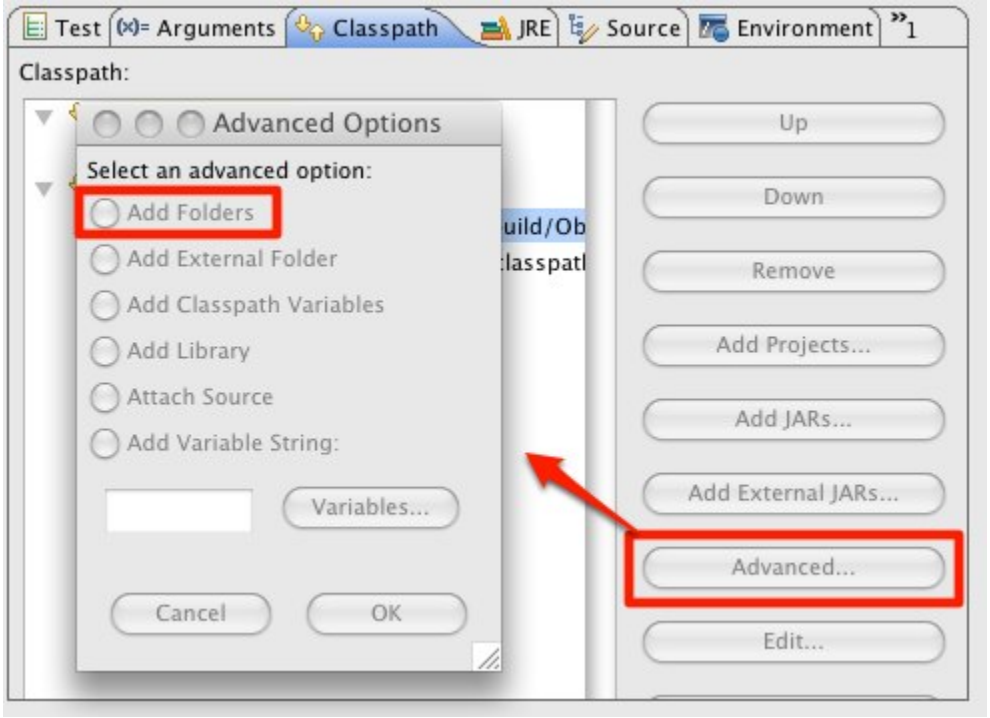
```
${workspace_loc:MyEclipseProject/build/MyEclipseProject.framework}
```

For an application it should be (using the handy wolips_dir_loc variable)

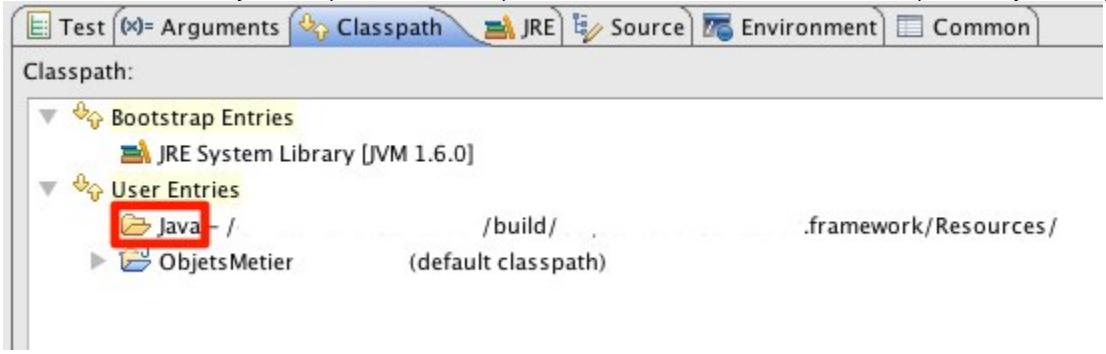
```
${working_dir_loc_WOLips:MyEcliseProject}
```



Then, in the classpath tab, select the *"User Entries"* entry and use the *"Advanced..."* button and select the *"Add Folders"* button.



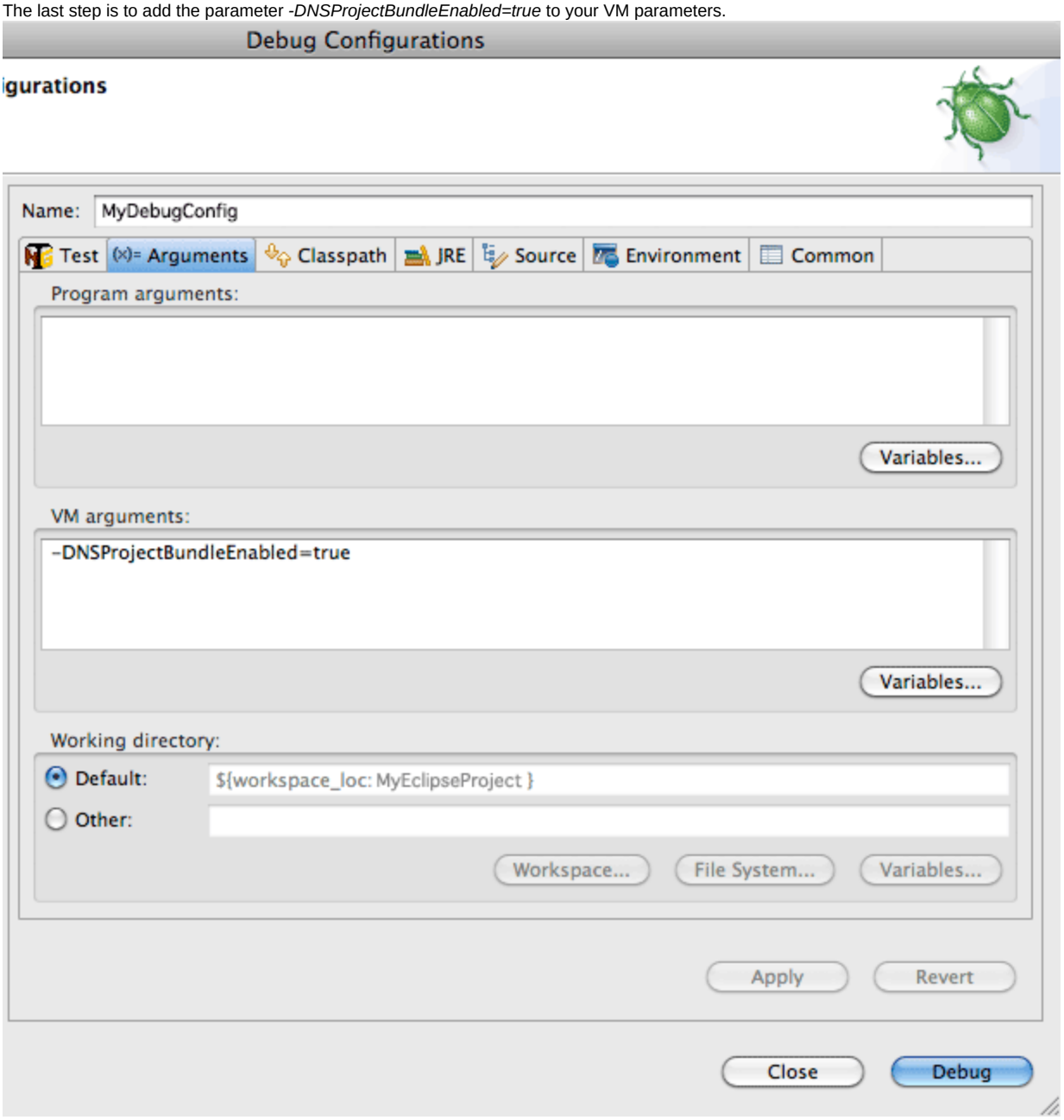
Select the *"Java"* folder of your build product. Use the *"Up/Down"* buttons to move this folder at the first position of your classpath.



Be Careful

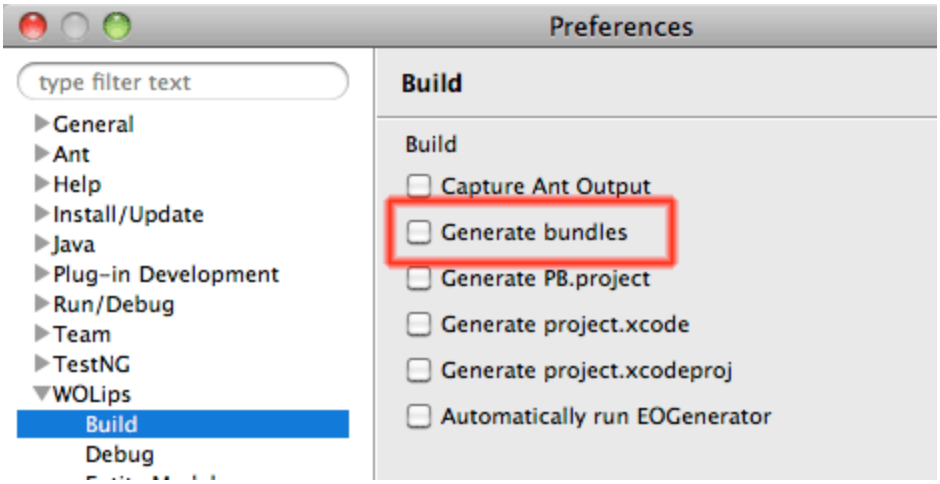
If during your tests execution you see error messages such as **cannot cast EOGenericRecord to <your object>** please check that the *"Java"* folder is at first position in your classpath entries.

The last step is to add the parameter *-DNSProjectBundleEnabled=true* to your VM parameters.



Bundle-less builds

A *bundle-less* build means that you have unchecked the build option *Generate bundles* within the WOLips preferences.



That means that there will be no *build* folder in your project. In that case you must set the working directoy to the default:

```
${workspace_loc:MyEclipseProject}
```

On the classpath tab you don't need to add the *"Java"* folder of your build product so only the default classpath should be listed for the *"User Entries"*. Be sure to add the parameter *-DNSProjectBundleEnabled=true* to your VM parameters as for *normal* builds.